Assignment 2: Contextualized Vectors, Parts of Speech, and Named Entities.

In the previous assignment, you worked with word vectors. In this assignment, we will move to contextualized (sentence dependent) vectors, such as those produced by sentence encoding models. We will do this while also considering the concepts of parts-of-speech tags and of named entities.

0) Warmup

The `huggingface transformers` python library provides many sentence encoders.
We will be using the “roberta-base” model.
https://huggingface.co/roberta-base

It was trained as a “masked language model”.

To learn to use the model, after installing the library (`pip install transformers`) do the following tasks:

0) Install the huggingface transformers library
1) Encode the sentence “I am so <mask>” and:
    1.1) extract the vectors for “am” and for “<mask>”.
    1.2) extract the top-5 word predictions for “am” and for “<mask>” and their probabilities.
2) Find two sentences that share the same word, such that the cosine similarity between the word vectors in the two sentences is **very high**.
3) Find two sentences that share the same word, such that the cosine similarity between the word vectors in the two sentences is **very low**.
4) Find a sentence with n words, that is tokenized into m > n tokens by the tokenizer.

1) Part-of-speech tagging

In this part of the assignment, we will explore the notion of part-of-speech tagging, that is, given a text, assign each word in the text its correct part-of-speech.

The “standard” way to perform part-of-speech tagging with word vectors, is to get a vector representation of each word (or of each word in its context) and then use an annotated corpus to *train a classifier* to predict the correct part-of-speech tag from the vector (or vectors). The classifier can either be on top of the vectors (where the encoder that produces the vectors is

fixed), or it may also include the encoder itself (so both the classifier and the encoder that produces the vectors are changed during the training process). Using such an approach, and based on the roberta-base encoder, we can easily get part-of-speech tagging accuracies of 97% or above. The `transformers` library, together with the `pytorch` library, makes such training quite easy, once you get the hang of it.

In this assignment, however, **we will not take this approach** (though you are certainly welcome and encouraged to experiment with it on your own, if you wish). Instead, we will experiment with predicting parts of speech of words in context based on the out-of-the-box vectors, without training any classifiers (that is, without changing any parameters, without running SGD, etc).

Training, validation (dev) and test data

The training, validation (dev) and test data are in the `pos.tgz` file.

The format is one sentence per line, the text is already pre-tokenized (in the sense of text tokenization that splits punctuation from words etc, not in the BPE/word-pieces sense that is needed to feed the sentence into a contextualized vectors encoder).

Task

In each of the following sections, your task will be simple: use the annotated data in the training set, to best predict the parts-of-speech of the words in the test set sentences.

You can do whatever you want to achieve a good accuracy, as long as you don't:

1) look at the labels of the test data—this is never allowed.
2) tune any parameters or train any model—this is not allowed in this assignment.

What can you do, then? You are free to choose any approach you wish. For example, you could encode some or all the data in the training set, remember the labels, and look for similarities to encoded test vectors. Or you could predict the top-k words for some or all the words in the training set, and use them somehow. Or you could use clustering of vectors. Or you could use dimensionality reduction as we did in assignment 1. Or you can mix-and-match these approaches. Or you could do something else.

Notes

Combining vectors: as we discussed in class, natural ways to combine vectors are by concatenation or by addition.

Runtime: we do not talk about efficiency or running time in this assignment, and things are possible to run also without a GPU. However, if your predictor cannot finish tagging the data before you need to submit the assignment, then there might be a problem.

What to submit: In addition to the written report, for each of the following subtasks, send a prediction file named `POS_preds_X.txt` where X is replaced by the subtask number (1, 2 or 3). Each line in the predictions should correspond to a test sentence and be in the same format of the training data.

1.1) No word vectors

In this part of the assignment, you are not allowed to use any word vectors at all. However, you are allowed to count word, count (word,tag) pairs, to look into the characters the words are made of and count them as well, etc.

One simple approach would be to count for each word type in the training corpus what is its most frequent part-of-speech tag, and then assign this tag for every occurrence of this word in the test corpus, regardless of its context (you also need to figure out which tags to assign to words that did not appear in the training corpus). How well does this approach get you? Can you do better? What is the accuracy of the best parts-of-speech predictor you can achieve with this approach? (reaching 85% accuracy should be trivial, and you can do more than that).

Describe your approach in the report, and submit your predictions over the test data

1.2) Static word vectors

As before, but now you are allowed to use static vectors (word2vec or glove vectors, as used in assignment 1). You can use either single vectors, or vectors in a window surrounding the word. You can also use some of the counts from part 1.1, if you find them helpful. What is the best accuracy you can get?

Describe your approach in the report, and submit your predictions over the test data

1.3) Contextualized word vectors

As before, but now you are allowed to use the output of roberta-base (either the word vectors for each position, or the word predictions for each position, or both, or any other output you can get out of the pre-trained roberta-base model). You can also use some of the counts from part 1.1, or the static vectors from part 1.2, if you find them helpful. What is the best accuracy you can get?

Describe your approach in the report, and submit your predictions over the test data.

2) Named Entities Recognition

We now switch from predicting parts-of-speech (which are per-word) to named entities (which are spans). As you will see in class, there is a reduction by which you can perform span prediction by predicting tags for individual words. This is also the format the annotated data comes in. However, you are not required to use this reduction in this assignment, you can also predict spans directly.

As before, the standard (and effective) way to perform named entity recognition, is to train (fine-tune) a classifier to predict the named-entity tags. You are, however, again, not allowed to do so in this assignment. Instead, you will need to work under the same constraints as in part 1.

You are allowed to use counts, as well as static word vectors, and the output of roberta-base (as in 1.3). What is the best named-entity-predictor you can produce?

Describe your approach in the report, and submit your predictions over the test data. The predictions should be sent in a text file `NER_preds.txt`. Here again each line should correspond to a test sentence and be in the same format of the input data.

Training, validation (dev) and test data

The training, validation (dev) and test data are in the `ner.tgz` file.
The format is like in the parts-of-speech part, using the BIO encoding for spans. Note however that while the input and output should be in the BIO format, your prediction algorithm could be different from treating the problem as a per-token tagging task, and can attempt to predict spans directly.

Evaluation metric

Your evaluation metrics in this part should be **precision, recall and F1 over predicted spans**, as learned in class. You can perform span-level evaluation using the `ner_eval.py` script, which should be run as:

```
python ner_eval.py gold_file predicted_file
```